Exercise 15

Binay P Jena

November 1st 2020

Exercise 15: Introduction to Machine Learning

Problem Statement : These assignments are here to provide you with an introduction to the "Data Science" use for these tools. This is your future. It may seem confusing and weird right now but it hopefully seems far less so than earlier in the semester. Attempt these homework assignments. You will not be graded on your answer but on your approach. This should be a, "Where am I on learning this stuff" check. If you can't get it done, please explain why.

Include all of your answers in a R Markdown report.

Regression algorithms are used to predict numeric quantity while classification algorithms predict categorical outcomes. A spam filter is an example use case for a classification algorithm. The input dataset is emails labeled as either spam (i.e. junk emails) or ham (i.e. good emails). The classification algorithm uses features extracted from the emails to learn which emails fall into which category.

In this problem, you will use the nearest neighbors algorithm to fit a model on two simplified datasets. The first dataset (found in binary-classifier-data.csv) contains three variables; label, x, and y. The label variable is either 0 or 1 and is the output we want to predict using the x and y variables. The second dataset (found in trinary-classifier-data.csv) is similar to the first dataset except that the label variable can be 0, 1, or 2.

Note that in real-world datasets, your labels are usually not numbers, but text-based descriptions of the categories (e.g. spam or ham). In practice, you will encode categorical variables into numeric values.

Solution

```
## Set the working directory to the root of your DSC 520 directory
setwd("/Users/binayprasannajena/Documents/GitHub/dsc520/")
## Load the `caTools` library
library(caTools)
## Load the `data/binary-classifier-data.csv` as df
binary_classifier_df <- read.csv("data/binary-classifier-data.csv")
head(binary_classifier_df)
```

```
##   label        x        y
## 1     0 70.88469 83.17702
## 2     0 74.97176 87.92922
## 3     0 73.78333 92.20325
## 4     0 66.40747 81.10617
## 5     0 69.07399 84.53739
## 6     0 72.23616 86.38403
```

```r
summary(binary_classifier_df)
```

```
##      label             x                y          
##  Min.   :0.000   Min.   : -5.20   Min.   : -4.019  
##  1st Qu.:0.000   1st Qu.: 19.77   1st Qu.: 21.207  
##  Median :0.000   Median : 41.76   Median : 44.632  
##  Mean   :0.488   Mean   : 45.07   Mean   : 45.011  
##  3rd Qu.:1.000   3rd Qu.: 66.39   3rd Qu.: 68.698  
##  Max.   :1.000   Max.   :104.58   Max.   :106.896
```

```r
## Load the 'data/trinary-classifier-data.csv' as df
trinary_classifier_df <- read.csv("data/trinary-classifier-data.csv")
head(binary_classifier_df)
```

```
##   label        x        y
## 1     0 70.88469 83.17702
## 2     0 74.97176 87.92922
## 3     0 73.78333 92.20325
## 4     0 66.40747 81.10617
## 5     0 69.07399 84.53739
## 6     0 72.23616 86.38403
```

```r
summary(binary_classifier_df)
```

```
##      label             x                y          
##  Min.   :0.000   Min.   : -5.20   Min.   : -4.019  
##  1st Qu.:0.000   1st Qu.: 19.77   1st Qu.: 21.207  
##  Median :0.000   Median : 41.76   Median : 44.632  
##  Mean   :0.488   Mean   : 45.07   Mean   : 45.011  
##  3rd Qu.:1.000   3rd Qu.: 66.39   3rd Qu.: 68.698  
##  Max.   :1.000   Max.   :104.58   Max.   :106.896
```

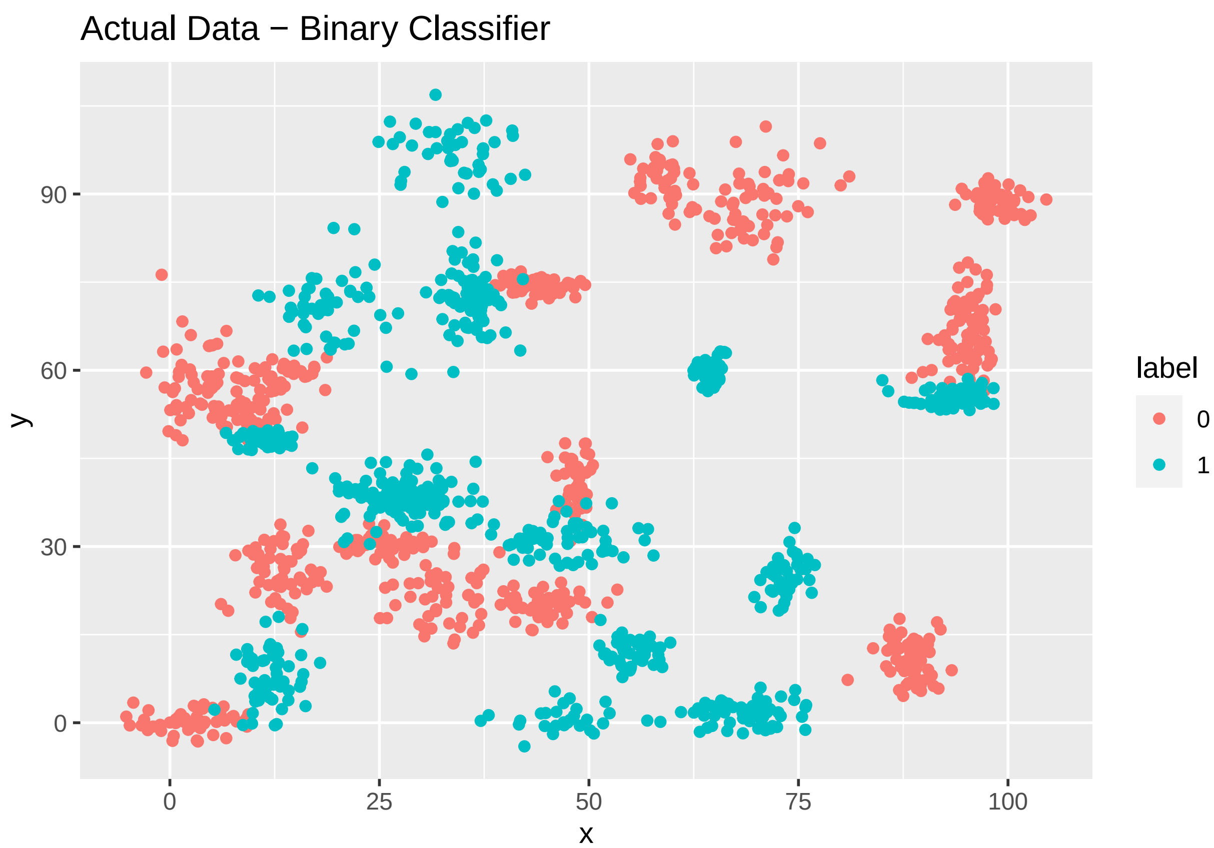
a. Plot the data from each dataset using a scatter plot.

Making Labels as Factor, since they are numbers

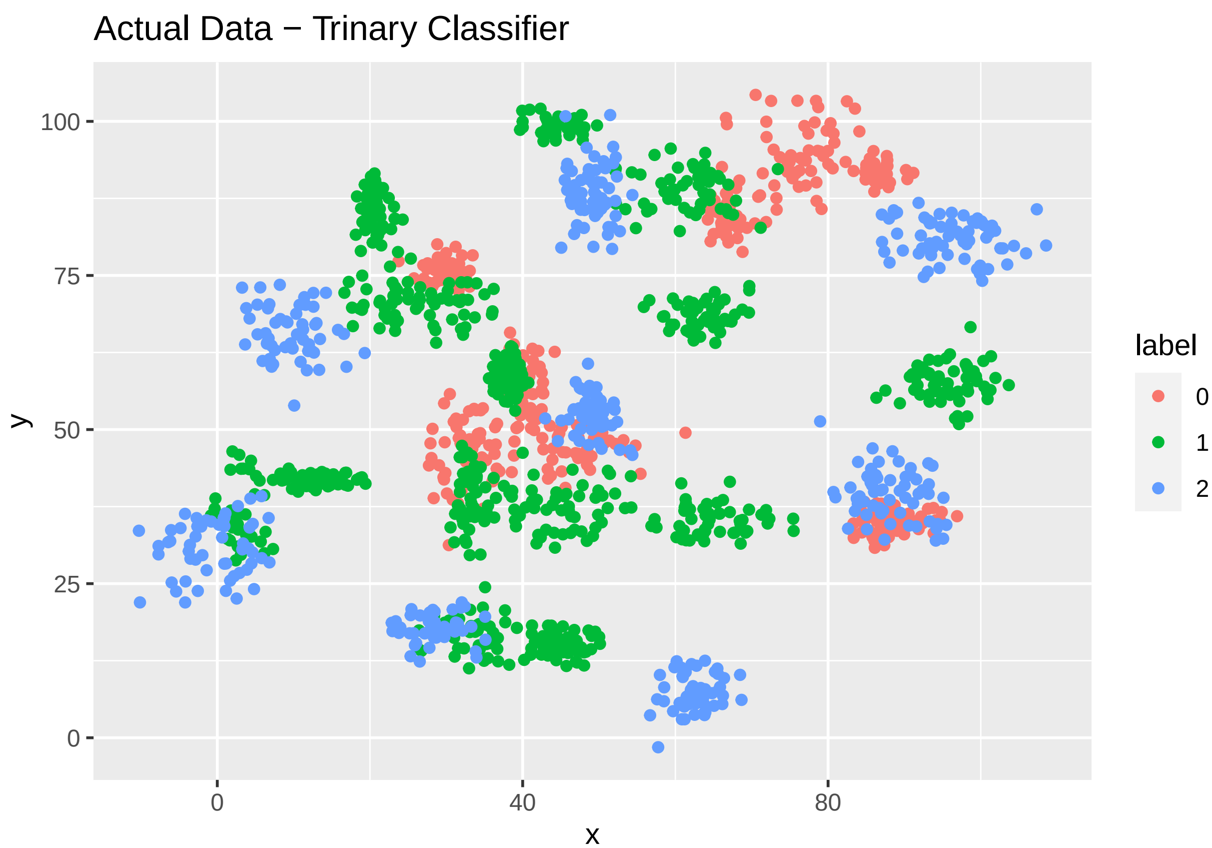
```r
# Since label is number converting to factors, so that it becomes categorical
binary_classifier_df$label <- as.factor(binary_classifier_df$label)
trinary_classifier_df$label <- as.factor(trinary_classifier_df$label)
```

Plotting

```r
library(ggplot2)
ggplot(data = binary_classifier_df, aes(y = y, x = x, color = label)) +
  geom_point() + ggtitle("Actual Data - Binary Classifier")
```




```r
ggplot(data = trinary_classifier_df, aes(y = y, x = x, color = label)) +
  geom_point() + ggtitle("Actual Data - Trinary Classifier")
```



b. The k nearest neighbors algorithm categorizes an input value by looking at the labels for the k nearest points and assigning a category based on the most common label. In this problem, you will determine which points are nearest by calculating the Euclidean distance between two points. As a refresher, the Euclidean distance between two points:p1=(x1, y1) and p2=(x2,y2) is d

Fitting a model is when you use the input data to create a predictive model. There are various metrics you can use to determine how well your model fits the data. You will learn more about these metrics in later lessons. For this problem, you will focus on a single metric; accuracy. Accuracy is simply the percentage of how often the model predicts the correct result. If the model always predicts the correct result, it is 100% accurate. If the model always predicts the incorrect result, it is 0% accurate.

Fit a k nearest neighbors model for each dataset for k=3, k=5, k=10, k=15, k=20, and k=25. Compute the accuracy of the resulting models for each value of k. Plot the results in a graph where the x-axis is the different values of k and the y-axis is the accuracy of the model.

Splitting the Data into Train and Test dataframes

```r
# Splitting the dataset for the model into train and test datasets.
myData_binary <- sample.split(binary_classifier_df$label, SplitRatio=0.8)
train_binary <- subset(binary_classifier_df, myData_binary==TRUE)
test_binary <- subset(binary_classifier_df, myData_binary==FALSE)
myData_trinary <- sample.split(trinary_classifier_df$label, SplitRatio=0.8)
train_trinary <- subset(trinary_classifier_df, myData_trinary==TRUE)
test_trinary <- subset(trinary_classifier_df, myData_trinary==FALSE)
```

Accuracy function

```r
# this function divides the correct predictions by total number of predictions that tell
# us how accurate the model is.
accuracy <- function(x){sum(diag(x)/(sum(rowSums(x)))) * 100}
#accuracy(tab)
```

Fitting K nearest Model for Binary Classifier

Generating KNN Models for Binary Classifier Data and Displaying their accuracy from K=1 to K=25

```r
library(class)
K_Binary_Values <- c()
Accuracy_Binary <- c()
error.rate.b <- c()
# Running for multiple K Values
for(i in 1:25){
  knnmodel.b.i <- knn(train_binary[2:3],test_binary[2:3],k=i,cl=train_binary$label)
  table.b.i <- table(knnmodel.b.i,test_binary$label)
  accur = accuracy(table.b.i)
  print(paste("Accuracy for Binary Classifier Data Model with K=", i ," is ", accuracy(table.b.i)))
  K_Binary_Values <- c(K_Binary_Values, i)
  Accuracy_Binary<- c(Accuracy_Binary, accur)
  error.rate.b <- c(error.rate.b, mean(test_binary$label != knnmodel.b.i))
}
```

```
## [1] "Accuracy for Binary Classifier Data Model with K= 1  is  96.6555183946488"
## [1] "Accuracy for Binary Classifier Data Model with K= 2  is  97.3244147157191"
## [1] "Accuracy for Binary Classifier Data Model with K= 3  is  97.6588628762542"
## [1] "Accuracy for Binary Classifier Data Model with K= 4  is  97.6588628762542"
## [1] "Accuracy for Binary Classifier Data Model with K= 5  is  96.989966555184"
## [1] "Accuracy for Binary Classifier Data Model with K= 6  is  96.989966555184"
## [1] "Accuracy for Binary Classifier Data Model with K= 7  is  96.989966555184"
## [1] "Accuracy for Binary Classifier Data Model with K= 8  is  97.3244147157191"
## [1] "Accuracy for Binary Classifier Data Model with K= 9  is  96.989966555184"
## [1] "Accuracy for Binary Classifier Data Model with K= 10  is  97.3244147157191"
## [1] "Accuracy for Binary Classifier Data Model with K= 11  is  97.3244147157191"
## [1] "Accuracy for Binary Classifier Data Model with K= 12  is  96.6555183946488"
## [1] "Accuracy for Binary Classifier Data Model with K= 13  is  96.6555183946488"
## [1] "Accuracy for Binary Classifier Data Model with K= 14  is  96.6555183946488"
## [1] "Accuracy for Binary Classifier Data Model with K= 15  is  96.6555183946488"
```

```
## [1] "Accuracy for Binary Classifier Data Model with K= 16  is  96.989966555184"
## [1] "Accuracy for Binary Classifier Data Model with K= 17  is  96.6555183946488"
## [1] "Accuracy for Binary Classifier Data Model with K= 18  is  96.989966555184"
## [1] "Accuracy for Binary Classifier Data Model with K= 19  is  96.989966555184"
## [1] "Accuracy for Binary Classifier Data Model with K= 20  is  96.989966555184"
## [1] "Accuracy for Binary Classifier Data Model with K= 21  is  97.3244147157191"
## [1] "Accuracy for Binary Classifier Data Model with K= 22  is  96.989966555184"
## [1] "Accuracy for Binary Classifier Data Model with K= 23  is  96.989966555184"
## [1] "Accuracy for Binary Classifier Data Model with K= 24  is  96.6555183946488"
## [1] "Accuracy for Binary Classifier Data Model with K= 25  is  96.6555183946488"
```

```
print(K_Binary_Values)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
```

```
print(Accuracy_Binary)
```

```
##  [1] 96.65552 97.32441 97.65886 97.65886 96.98997 96.98997 96.98997 97.32441
##  [9] 96.98997 97.32441 97.32441 96.65552 96.65552 96.65552 96.65552 96.98997
## [17] 96.65552 96.98997 96.98997 96.98997 97.32441 96.98997 96.98997 96.65552
## [25] 96.65552
```

```
print(error.rate.b*100)
```

```
##  [1] 3.344482 2.675585 2.341137 2.341137 3.010033 3.010033 3.010033 2.675585
##  [9] 3.010033 2.675585 2.675585 3.344482 3.344482 3.344482 3.344482 3.010033
## [17] 3.344482 3.010033 3.010033 3.010033 2.675585 3.010033 3.010033 3.344482
## [25] 3.344482
```

```
print(Accuracy_Binary+error.rate.b*100)
```

```
##  [1] 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100
## [20] 100 100 100 100 100 100
```

```
accuracy_binary_df <- data.frame(K_Binary_Values, Accuracy_Binary, error.rate.b*100)
accuracy_binary_df
```

```
##    K_Binary_Values Accuracy_Binary error.rate.b...100
## 1                1        96.65552           3.344482
## 2                2        97.32441           2.675585
## 3                3        97.65886           2.341137
## 4                4        97.65886           2.341137
## 5                5        96.98997           3.010033
## 6                6        96.98997           3.010033
## 7                7        96.98997           3.010033
## 8                8        97.32441           2.675585
## 9                9        96.98997           3.010033
## 10              10        97.32441           2.675585
## 11              11        97.32441           2.675585
## 12              12        96.65552           3.344482
```

```
## 13               13          96.65552            3.344482
## 14               14          96.65552            3.344482
## 15               15          96.65552            3.344482
## 16               16          96.98997            3.010033
## 17               17          96.65552            3.344482
## 18               18          96.98997            3.010033
## 19               19          96.98997            3.010033
## 20               20          96.98997            3.010033
## 21               21          97.32441            2.675585
## 22               22          96.98997            3.010033
## 23               23          96.98997            3.010033
## 24               24          96.65552            3.344482
## 25               25          96.65552            3.344482
```

Fitting K nearest Model for Trinary Classifier

Generating KNN Models for Trinary Classifier Data and Displaying their accuracy from K=1 to K=25

```r
K_Trinary_Values <- c()
Accuracy_Trinary <- c()
error.rate.t <- c()
# Running for multiple K Values
for(i in 1:25){
  knnmodel.t.i <- knn(train_trinary[2:3],test_trinary[2:3],k=i,cl=train_trinary$label)
  table.t.i <- table(knnmodel.t.i,test_trinary$label)
  accur = accuracy(table.t.i)
  print(paste("Accuracy for Trinary Classifier Data Model with K=", i ," is ", accur))
  K_Trinary_Values <- c(K_Trinary_Values, i)
  Accuracy_Trinary<- c(Accuracy_Trinary, accur)
  error.rate.t <- c(error.rate.t, mean(test_trinary$label != knnmodel.t.i))
}
```

```
## [1] "Accuracy for Trinary Classifier Data Model with K= 1  is  85.9424920127796"
## [1] "Accuracy for Trinary Classifier Data Model with K= 2  is  84.0255591054313"
## [1] "Accuracy for Trinary Classifier Data Model with K= 3  is  87.5399361022364"
## [1] "Accuracy for Trinary Classifier Data Model with K= 4  is  86.2619808306709"
## [1] "Accuracy for Trinary Classifier Data Model with K= 5  is  87.8594249201278"
## [1] "Accuracy for Trinary Classifier Data Model with K= 6  is  87.5399361022364"
## [1] "Accuracy for Trinary Classifier Data Model with K= 7  is  89.1373801916933"
## [1] "Accuracy for Trinary Classifier Data Model with K= 8  is  90.0958466453674"
## [1] "Accuracy for Trinary Classifier Data Model with K= 9  is  89.1373801916933"
## [1] "Accuracy for Trinary Classifier Data Model with K= 10  is  88.8178913738019"
## [1] "Accuracy for Trinary Classifier Data Model with K= 11  is  89.4568690095847"
## [1] "Accuracy for Trinary Classifier Data Model with K= 12  is  89.4568690095847"
## [1] "Accuracy for Trinary Classifier Data Model with K= 13  is  89.776357827476"
## [1] "Accuracy for Trinary Classifier Data Model with K= 14  is  88.8178913738019"
## [1] "Accuracy for Trinary Classifier Data Model with K= 15  is  89.4568690095847"
## [1] "Accuracy for Trinary Classifier Data Model with K= 16  is  88.8178913738019"
## [1] "Accuracy for Trinary Classifier Data Model with K= 17  is  89.1373801916933"
## [1] "Accuracy for Trinary Classifier Data Model with K= 18  is  89.1373801916933"
## [1] "Accuracy for Trinary Classifier Data Model with K= 19  is  87.8594249201278"
## [1] "Accuracy for Trinary Classifier Data Model with K= 20  is  88.4984025559105"
## [1] "Accuracy for Trinary Classifier Data Model with K= 21  is  87.8594249201278"
```

```
## [1] "Accuracy for Trinary Classifier Data Model with K= 22  is  87.8594249201278"
## [1] "Accuracy for Trinary Classifier Data Model with K= 23  is  89.4568690095847"
## [1] "Accuracy for Trinary Classifier Data Model with K= 24  is  88.1789137380192"
## [1] "Accuracy for Trinary Classifier Data Model with K= 25  is  89.776357827476"
```

```
print(K_Trinary_Values)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
```

```
print(Accuracy_Trinary)
```

```
##  [1] 85.94249 84.02556 87.53994 86.26198 87.85942 87.53994 89.13738 90.09585
##  [9] 89.13738 88.81789 89.45687 89.45687 89.77636 88.81789 89.45687 88.81789
## [17] 89.13738 89.13738 87.85942 88.49840 87.85942 87.85942 89.45687 88.17891
## [25] 89.77636
```

```
print(error.rate.t*100)
```

```
##  [1] 14.057508 15.974441 12.460064 13.738019 12.140575 12.460064 10.862620
##  [8]  9.904153 10.862620 11.182109 10.543131 10.543131 10.223642 11.182109
## [15] 10.543131 11.182109 10.862620 10.862620 12.140575 11.501597 12.140575
## [22] 12.140575 10.543131 11.821086 10.223642
```

```
print(Accuracy_Trinary+error.rate.t*100)
```

```
##  [1] 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100
## [20] 100 100 100 100 100 100
```

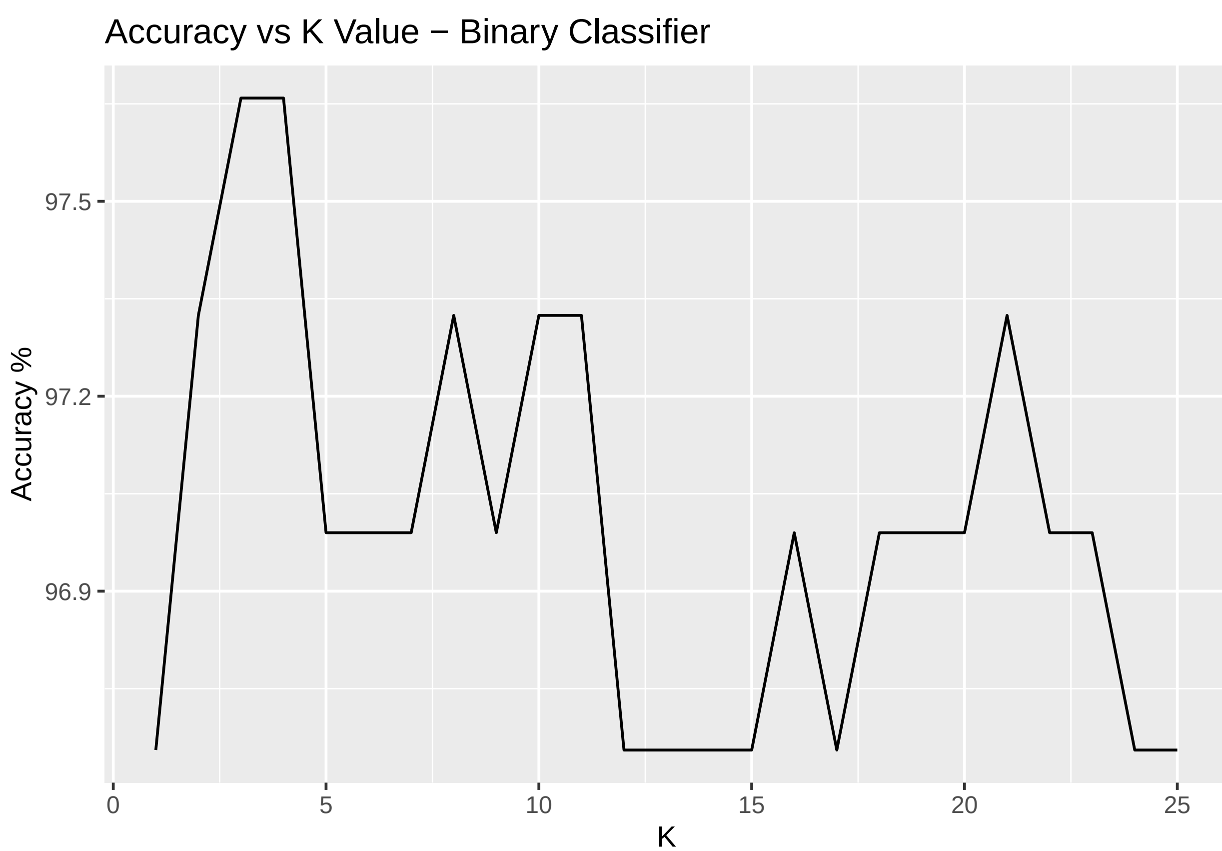
```
accuracy_trinary_df <- data.frame(K_Trinary_Values, Accuracy_Trinary, error.rate.t*100)
accuracy_trinary_df
```

```
##    K_Trinary_Values Accuracy_Trinary error.rate.t...100
## 1                 1         85.94249          14.057508
## 2                 2         84.02556          15.974441
## 3                 3         87.53994          12.460064
## 4                 4         86.26198          13.738019
## 5                 5         87.85942          12.140575
## 6                 6         87.53994          12.460064
## 7                 7         89.13738          10.862620
## 8                 8         90.09585           9.904153
## 9                 9         89.13738          10.862620
## 10               10         88.81789          11.182109
## 11               11         89.45687          10.543131
## 12               12         89.45687          10.543131
## 13               13         89.77636          10.223642
## 14               14         88.81789          11.182109
## 15               15         89.45687          10.543131
## 16               16         88.81789          11.182109
## 17               17         89.13738          10.862620
## 18               18         89.13738          10.862620
```

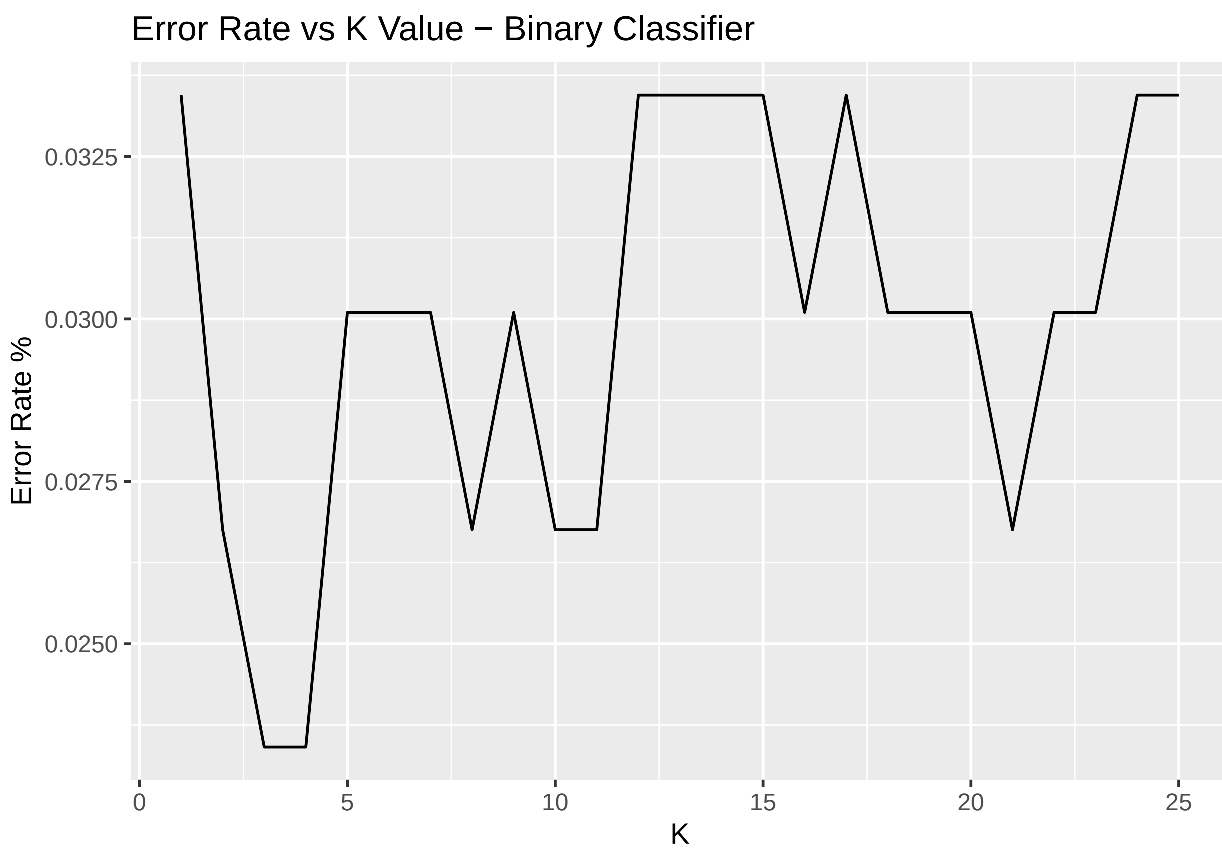
```
## 19                19          87.85942          12.140575
## 20                20          88.49840          11.501597
## 21                21          87.85942          12.140575
## 22                22          87.85942          12.140575
## 23                23          89.45687          10.543131
## 24                24          88.17891          11.821086
## 25                25          89.77636          10.223642
```

Plotting the K Values vs Accuracy

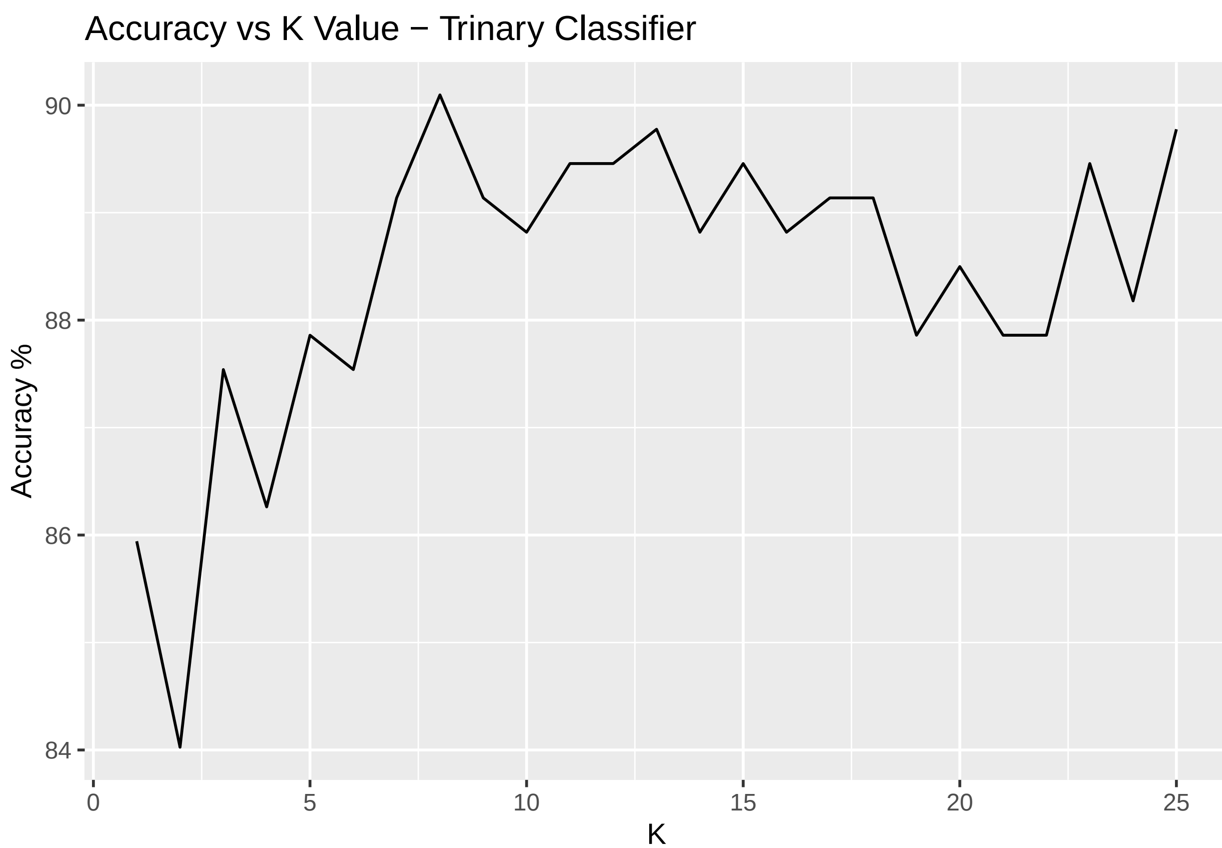
```
library(ggplot2)
ggplot(data = accuracy_binary_df, aes(y = Accuracy_Binary, x = K_Binary_Values)) +
  geom_line() + ggtitle("Accuracy vs K Value - Binary Classifier") +
  ylab("Accuracy %") + xlab("K")
```



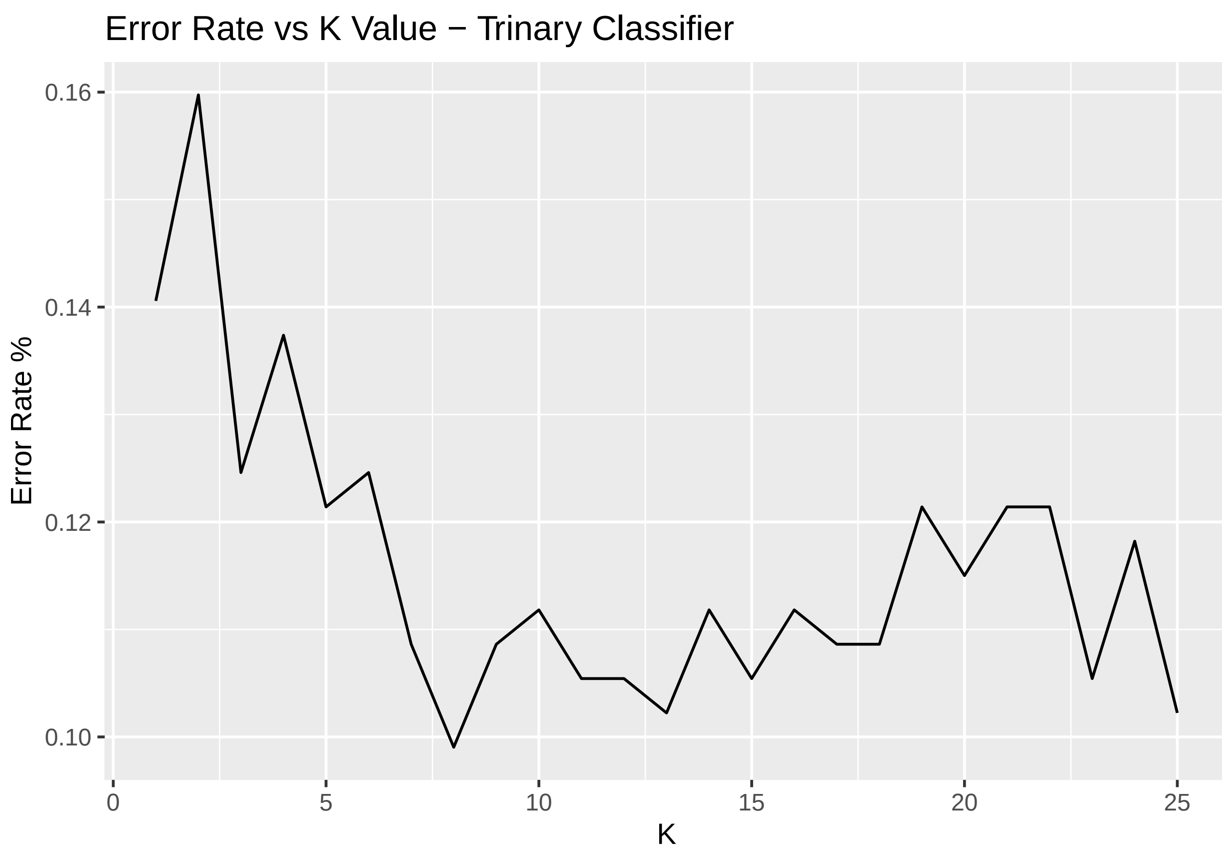

```
# Inverted Plot of Above
ggplot(data = accuracy_binary_df, aes(y = error.rate.b, x = K_Binary_Values)) +
  geom_line() + ggtitle("Error Rate vs K Value - Binary Classifier") +
  ylab("Error Rate %") + xlab("K")
```




```r
ggplot(data = accuracy_trinary_df, aes(y = Accuracy_Trinary, x = K_Trinary_Values)) +
  geom_line() + ggtitle("Accuracy vs K Value - Trinary Classifier") +
  ylab("Accuracy %") + xlab("K")
```




```r
# Inverted Plot of Above
ggplot(data = accuracy_trinary_df, aes(y = error.rate.t, x = K_Trinary_Values)) +
  geom_line() + ggtitle("Error Rate vs K Value - Trinary Classifier") +
  ylab("Error Rate %") + xlab("K")
```



c. In later lessons, you will learn about linear classifiers. These algorithms work by defining a decision boundary that separates the different categories. Looking back at the plots of the data, do you think a linear classifier would work well on these datasets?

If we look at the Actual Data, its hard to divide the data(colored dots) with a line / linear classifier as the data is spread in different clusters. It is not possible to put the different binary values in opposite direction of this line, So linear classifier will not work with this kind of dataset.

Reference

https://towardsdatascience.com/k-nearest-neighbors-algorithm-with-examples-in-r-simply-explained-knn-1f2c88da405c